CERTIFICATE OF MAILING BY EXPRESS MAIL

"EXPRESS MAIL" Mailing Label No. EV226957205US

Date of Deposit: SEPTEMBER 19, 2003
I hereby certify that this paper or fee is being deposited with the U.S. Postal Service "Express Mail Post Office to Addressee" service on the date indicated above and is addressed to the Commissioner for Patents, Mail Stop Patent Application, P.O. Box 1450, Alexandria, VA 22313-1450

Type or Print Name: CAROL MITCHELL

Carol Mitchell
Signature

Title: THE PLUG-IN MODEL

Inventor(s):
1. Björn BJÄRE
2. Chi Thu LE
3. Jonas HANSSON
4. Henrik SVENSSON
5. Mikael DANIELSSON

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims the benefit of priority from and incorporates by reference the entire disclosure of co-pending U.S. Provisional Patent Application Serial No. 60/412,902, filed September 23, 2002 and bearing Attorney Docket No. 53807-00057USPL. This application claims the benefit of priority from and incorporates by reference the entire disclosure of co-pending U.S. Provisional Patent Application Serial No. 60/412,901, filed September 23, 2002 and bearing Attorney Docket No. 53807-00050USPL. This application claims the benefit of priority from and incorporates by reference the entire disclosure of co-pending U.S. Provisional Patent Application Serial No. 60/412,769, filed September 23, 2002 and bearing Attorney Docket No. 53807-00055USPL. This application claims the benefit of priority from and incorporates by reference the entire disclosure of co-pending U.S. Provisional Patent Application Serial No. 60/412,756, filed September 23, 2002 and bearing Attorney Docket No. 53807-00059USPL. This patent application incorporates by reference the entire disclosure of U.S. Patent Application No. 10/359,772, which was filed on February 7, 2003 and bears Attorney Docket No. 53807-00024USPT. This patent application incorporates by reference the entire disclosure of U.S. Patent Application No. 10/359,835, which was filed on February 7, 2003 and bears Attorney Docket No. 53807-00045USPT. This patent application incorporates by reference the entire disclosure of U.S. Patent Application No. 10/359,911, which was filed on February 7, 2003 and bears Attorney Docket No. 53807-00023USPT.

BACKGROUND OF THE INVENTION

Technical Field of the Invention

The present invention relates generally to the field of wireless telecommunications; more particularly, the present invention relates to a method and system for extending and modifying the functionality of a platform for a mobile terminal for a wireless telecommunications system, without modifying the base functionality.

Description of Related Art

A platform that provides a high degree of configurability is an attractive option for many manufacturers of end user mobile products and other platform customers. Configurability is especially important with respect to high level functionality in that some customers desire such functionality, and often prefer to develop such functionality in-house, while others have no interest in including such functionality in their products.

A mobile terminal platform that provides a high degree of configurability, however, introduces a number of difficulties for the provider of the platform. For example, the provider must support any high-level functionality that is added to the platform and support the numerous configurations that different customers might require, such as by providing customers with platform documentation that is specific for their particular configuration. Also, since not all configurations of a platform can be expected to operate properly, it becomes necessary for the platform provider to test each specific configuration to ensure that the added functionality works in a manner desired by the customer. In addition, in many cases, the added functionality will be dependent on and have logical dependencies towards other functionality that is provided in the platform, such that if the latter functionality is removed, the added functionality may no longer operate properly.

Furthermore, it is frequently the case that a platform provider wishes to maintain details of its platform proprietary. To achieve this, the platform provider should not expose the code base directly to a customer or to any other party. Instead, it is preferable that a customer views the platform simply as a "black box" having a defined interface and defined behavior.

A platform having a high degree of configurability, however, can present difficulties with respect to maintaining details of the platform proprietary. For example, it is a common practice for a platform provider to contract a third party to develop high level add-ons to a platform. The platform provider might develop applications for the platform that are delivered with the platform. A typical example where this might occur is with respect to product-specific functionality such as functionality that controls the look and feel of the user interface, which is often implemented as a high-level graphics library. Since the look and feel is brand-specific, this functionality will differ among products and brands.

If a third-party is contracted to develop high-level add-ons, however, that party will require access to the platform code base in order to develop the functionality, forcing the provider to reveal proprietary information to the third party.

In addition, after a platform has been configured and delivered to a customer, the functionality of the platform will have been tested, stabilized and fixed. The customer, however, may have additional requirements with respect to modifying, extending and configuring the platform that are precluded by the design of the delivered platform. For example, a customer might wish to modify various parts of the provided functionality to tailor the platform to particular needs or to add more powerful functionality than is included in a delivered platform, i.e., functionality that the platform provider considers to be outside the scope of the platform domain. In addition, a customer might want to configure the platform functionality on a more detailed level than is offered by the platform provider in the basic platform assembly configuration, or to perform some form of last minute configuration to add or remove functionality to the platform based on what an end user is prepared to pay.

In general, current platforms for mobile terminals and other products do not provide an effective procedure by which the functionality of the platform can be extended, removed, or otherwise changed.

SUMMARY OF THE INVENTION

The present invention provides a method and system by which the functionality of a platform for a mobile terminal for a wireless telecommunications system or for another product can be extended or otherwise changed. A system for extending and/or modifying functionality of a platform for a product includes a platform domain having a software services component for providing functionality and an interface component having at least one interface for providing access to the functionality of the software services component for enabling application domain software to be installed, loaded and run in said platform via said at least one interface. The system also includes plug-in software for use by the application software for extending and/or modifying the functionality of the software services component of the platform domain via the at least one interface.

A method for extending and/or modifying functionality of a platform for a product includes providing a platform domain having a software services component for providing functionality, and an interface component having at least one interface for providing access to the functionality of the software services component for enabling application domain software to be installed, loaded and run in said platform via said at least one interface. The method also includes providing plug-in software together with the application software for extending and/or modifying the functionality of the software services component of the

platform domain via the at least one interface. The method also includes extending and/or modifying the functionality of the software services component via said plug-in software.

Further advantages and specific details of the present invention will become apparent hereinafter from the detailed description given below in conjunction with the following drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 is a block diagram that schematically illustrates a platform system for a mobile terminal for a wireless telecommunications system to assist in explaining principles of the present invention;

FIGURE 2 is a block diagram that schematically illustrates a deployment view of the mobile terminal platform assembly of the platform system of FIGURE 1 to further assist in explaining principles of the present invention;

FIGURE 3 is a block diagram that schematically illustrates the software architecture of the mobile terminal platform assembly of FIGURES 1 and 2 to further assist in explaining principles of the present invention;

FIGURE 4A is a logical block diagram that schematically illustrates details of the middleware services layer of FIGURES 1-3;

FIGURE 4B is an implementation view of the block diagram of FIGURE 4A;

FIGURE 5 is a block diagram that schematically illustrates major software modules of the Open Platform API (OPA) domain;

FIGURES 6A and 6B are block diagrams that schematically illustrate logical and implementation views, respectively, of an Open Platform API (OPA) domain; and

FIGURES 7A and 7B are diagrams that schematically illustrate sequences for a plug-in model to accomplish message model compliance in a callback mode and a full message mode, respectively.

DETAILED DESCRIPTION OF THE EXEMPLARY EMBODIMENTS OF THE INVENTION

FIGURE 1 is a block diagram that schematically illustrates a platform system for a mobile terminal for a wireless telecommunications system to assist in explaining principles of the present invention. The platform system is generally designated by reference number 10 and includes a mobile terminal platform assembly 12 and one or more applications (i.e., application software) 14 that have been installed, loaded and run in the mobile terminal platform assembly. Platform system 10 is adapted to be incorporated in a mobile terminal generally designated by dotted line 16. Mobile terminal platform assembly 12 includes a software services component 22, a hardware component 24, and an interface component 26. Software services component 22 includes a plurality of well-structured functional software units for providing services that are offered to users via the interface component 26. In the exemplary system 10 illustrated in FIGURE 1, the plurality of software units include a plurality of vertically-oriented functional software stacks 30-38.

In the exemplary system 10 illustrated in FIGURE 1, the hardware component 24 includes a set of hardware blocks 40-48 that are associated with and controlled by their respective functional software stacks 30-38. The interface component 26 includes a middleware services layer that includes at least one application programming interface (API) for installing, loading and running one or more applications 14 in mobile terminal platform assembly 12, that isolates the mobile terminal platform assembly 12 from the applications 14 using the mobile terminal platform assembly 12, and that provides various other services for the applications 14.

Mobile terminal platform assembly 12 of platform system 10 is adapted to be designed, implemented (assembled) and tested as a complete, enclosed unit separate from the

application software 14 (the term "application software" as used herein can be any software that provides the functionality that users may wish to have available). Users can, accordingly, develop or otherwise acquire their own application software and add that software to the mobile terminal platform assembly 12 at a later time in order to complete platform system 10. Mobile terminal platform assembly 12 can, accordingly, be sold or otherwise transferred to a plurality of different users each of which can complete platform system 10 by installing, loading and running their own application software in the assembly in order to satisfy their own particular requirements for the platform system.

FIGURE 2 is a block diagram that schematically illustrates one example of a deployment view of mobile terminal platform system 12 of FIGURE 1 to further assist in understanding principles of the present invention. As illustrated in FIGURE 2, mobile terminal platform assembly 12 is controlled via software executing in a main CPU 50. The main CPU 50 may include one or more processors such as microprocessors, micro programmable processors or DSPs (Digital Signal Processors). The software stacks 30-38 of software component 22 each include hardware driver software 60-68 to operate the hardware units associated with each stack. Further details of the mobile terminal platform assembly 12 and platform system 10 are given in commonly assigned U.S. Patent Application Serial No. 10/359,835, the disclosure of which is hereby incorporated by reference.

The software incorporated in mobile terminal platform assembly 12 is preferably arranged in such a manner as to make the software organization easy to understand so that it can be more easily designed and more easily upgraded or otherwise modified. FIGURE 3 is a block diagram that schematically illustrates the software architecture of mobile terminal platform assembly 12 to further assist in explaining principles of the present invention.

As shown in FIGURE 3, software services component 22, in addition to being organized into a plurality of vertical, functional software stacks as described above, is also arranged to define a plurality of horizontal layers such that the software of the middleware services layer and the software of the software services component 22 together define a layered architecture, generally designated by reference number 70, in which the layers are arranged in descending order from a higher level service layer to a lower level service layer.

The software architecture differs from the standard ISO/OSI (ISO Open Systems Interconnection) model in that it includes a plurality of horizontally partitioned functional software units that complement a plurality of vertically partitioned software layers. The horizontal partitioning contributes significantly to the creation of independent modular (service) components.

The highest layer of the layered architecture is the middleware services layer, which is part of the interface component 26. The layers of the software services component 22 include an application service layer 80 to provide application services, a platform services layer 82 to provide platform specific services for applications, a platform protocol layer 84 to provide session protocols and application specific protocols, a transport layer 86 to provide audio access/control, datacom transport protocols, messaging transport protocols and the like, a data access layer 88 to provide external data IF access, structured storage services and other low level platform support services, a logical drivers layer 90 and a physical drivers layer 92 encapsulating hardware dependencies. In addition, software services component 22 includes basic system services layers 94 that provide general services that are needed by the platform assembly 12.

The bottom two layers 90 and 92 constitute Hardware Abstraction Layers (HAL) which isolate the dependencies between the software and the hardware. Only the physical

drivers layer is concerned with the details of the hardware (i.e., which registers in the ASIC hardware are addressed). The logical drivers layer 90 provides a logical mapping to the hardware, i.e., this layer provides a bridge between the hardware and software parts of the mobile terminal platform assembly.

The software itself is organized into a plurality of software modules, modules 102, 104, 106 being specifically indicated in FIG. 3. In software services component 22, a single module can reside in only one vertical functional stack and in only one horizontal layer within that stack. Each layer can contain from one to many modules, and all the modules in a particular layer in a particular stack have the same level of abstraction. Communication among the various modules is accomplished via a Software Back Plane (SwBP) 112 subject to a set of basic rules for software module-to-module access. These rules can be summarized as follows:

- A software module may invoke functionality in all layer interfaces below its own layer.
- There are no limitations for the direction of serialized data flows holding e.g. event information or data streams. They may go in any direction.
- A software module may never invoke functionality in layer interfaces (in the SwBP 112) above its own layer, independent of to which module the layers belong.
- A software module may invoke functionality in the layer interface in its own layer in the same vertical stack.

- A software module may invoke functionality in a software module in the same layer in another vertical stack. (This capability is permitted to limit the number of layers in the vertical stacks.)

There is no hard coupling between the various modules and the interfaces in the SwBP 112. As a result, the modules and/or the implementation of the interfaces can be freely changed without any impact on the clients to the interfaces. This capability permits individual modules to be added, removed or changed without affecting other modules in the platform assembly. Further details of the layered architecture, including the SwBP software structure that enables the internal communication between modules within the mobile terminal platform assembly are described in commonly assigned, U.S. Patent Application No. 10/359,911, the disclosure of which is hereby incorporated by reference.

Middleware services layer functions to provide a well-defined interface between the software in the mobile terminal platform assembly 12 and the application software 14 to be installed, loaded and run in the platform assembly 12; and, in addition, encapsulates the mobile terminal platform assembly 12 and isolates the assembly 12 from applications via the middleware services layer, and provides various other services for the applications 14.

FIGURE 4A is a logical block diagram that schematically illustrates details of the middleware services layer of the interface component 26. As shown in FIGURE 4A, middleware services layer includes a plurality of API domains including non-native environments e.g. Java Execution (Java ExE) Environment domain 202, Open Application Framework (OAF) domain 204, Open Platform API (OPA) domain 206 and UI Tool-kit domain 208.

Through the APIs in the middleware services layer, the mobile terminal platform assembly 12 supports a plurality of application environments. In the exemplary embodiment of FIGURE 4A, middleware services layer supports environments for native applications (applications that are compiled to run with a particular processor and its set of instructions) and for non-native applications such as Java J2ME CLDC/MIDP (Java 2 Micro Edition Connected Limited Device Configuration/Mobile Information Device Profile) applications. Each execution environment has its own characteristics and is defined as:

-- The way applications are developed (programming language support, compilation and linkage).
-- The way applications are executed (e.g., interpretation or native code execution)
-- The functional services that are offered.
-- Potential restrictions in use.

By providing multiple application environment alternatives, a wide range of products with varying demands such as cost, ease of use, time to market, functionality set, size, portability, etc. is facilitated. Each of the API domains includes a plurality of software modules, and details of various of the domains are described in commonly assigned U.S. Patent Application Serial No. 10/359,772, the disclosure of which is hereby incorporated by reference.

FIGURE 4B is an implementation view of an Application domain 500 and a Middleware domain 501. It illustrates two Plugins, Plugin A 510 and Plugin B 512, both of which reside in the Application domain 500 in order to extend and/or modify the services of an Open Platform API 514 of the Middleware domain 501. The Plugin A 510 may depend

on the Open Platform API 514 only, while the Plugin B 512 may depend on the Plugin A 510 as well as the Open Platform API 514. Both native applications 508 and non-native applications 506 may depend on the Plugin A 510, on the Plugin B 512, as well as on the Open Platform API (OPA) 514. In case of a non-native environment (e.g., a Java Virtual Machine), the application dependency is indirect, in the sense that the applications 506 depend on a non-native environment 504. The non-native environment 504 depends on the Plugin A 512, the Plugin B 514 or on the Open Platform API 514.

FIGURE 5 is a block diagram that schematically illustrates the major software modules of the Open Platform API (OPA) domain 206 according to an exemplary embodiment of the present invention. As illustrated, the OPA domain 206 includes five modules: a Native Environment Management (NEM) module 230, a Native Application Core (NAC) module 232, an OPA Interface and Handlers module 234, a Middleware Support Services module 236 and a Native Extension plug-in module(s) 238.

The Native Environment Management module 230 has the responsibility of controlling native applications in platform system 10. It is the recipient of the control commands concerning native applications, and keeps track of native applications that are currently running in the system.

The Native Application Core module 232 administers and takes care of the threading and message-handling complexities that the applications would otherwise have to handle themselves. The NAC module 232 also serves the purpose of achieving OS independence by simplifying the implementation details of the OS for relieving applications from run-time complexities, including message routing/filtering and message-related resource handling. Yet another important responsibility of the Native Application Core is to simplify the details of the start-up and shutdown phase of an application and in the handling of messages.

The Middleware Support Services module 236 provides services to the OPA domain that are common for the different handlers or that need to be centralized, e.g., object management and resource supervision.

The Native Extension plug-in module(s) 238 provides a flexible extension possibility for the provider of platform assembly 12. The plug-in, in effect extends the functionality of the platform assembly by providing additional interfaces and additional services to the application software 14 written by an end user equipment manufacturer or another party.

While FIGURE 5 is a logical view that illustrates the plug-in as a module of the Open Platform API (OPA) domain 206, FIGURE 6A is a logical view illustrating a more general representation of an Open Platform API (OPA) domain 240 having a plurality of Open Platform API modules 242, 244, 246 and 248, and a plug-in module 250 according to another exemplary embodiment of the present invention. In this logical view, plug-in 250 is also shown as being incorporated in Open Platform API 240 because, from the viewpoint of a customer, plug-in 250 will behave and appear as part of the Open Platform API in the middleware services component 260 of the platform assembly. In actuality, however, plug-in 250 includes a stand-alone entity running in the application software domain 270 on top of platform assembly 12, and that uses the functionality that is provided by the platform assembly. Thus, from an implementation point of view, it is located above the platform assembly, using the Open Platform API to access the platform assembly functionality as illustrated in the implementation view of FIGURE 6B

The plug-in supports all of the application software mechanisms supported by the platform assembly, and complies with the application model defined in the platform assembly. In other words, the plug-in will adhere to the same paradigm(s) as the provided OPA services exported by the Open Platform API. These include, but are not limited to:

Component model compliance in terms of how interfaces and components are defined and operate. The plug-in is implemented as a component and will provide its services to end-product application software through a defined function or method based interface.

- Naming convention compliance. The plug-in uses the same naming convention for the interface methods as the Open Platform API. Also both the parameters and types defined in these interface methods will comply with the naming conventions used in the Open Platform API.
- Undesired-event handling compliance. The plug-in will handle possible erroneous behavior of the functionality in the same way as the platform. Information about such errors will be delivered to the application software according to the same paradigm as in the Open Platform API.
- Message model compliance. The plug-in will support two modes for delivering results from asynchronous requests, call back, and full message mode services.
- object and interface based paradigm

With respect to the next-to-last point mentioned above, message model compliance has a major effect on the behavior of the plug-in and on how the plug-in is constructed. In particular, the platform assembly message model offers dual modes for the application software for the purpose of receiving results from asynchronous service requests and resulting from event subscriptions. The dual modes imply that the result is received, either via a callback mechanism (callback mode), or via a message queue (full message mode). This provides a flexibility and freedom for the platform assembly customer software developer to structure different applications according to different message models, depending on the nature of the application functionality.

The application determines the message mode in which the application will receive an asynchronous result by executing, or not executing, a return statement. If the application decides to receive the result in callback mode, the application returns execution control after having completed an asynchronous service request. If the decision is to go for the full message mode, the application does not return execution control after the service request, but rather polls the message queue and invoke the message-handling code manually by itself. Another benefit of this solution is that an application may actually change between the callback and full message mode at any point in runtime, in case this is needed or desirable for a certain problem or context.

The plug-in model complies with the dual modes of the message model, and will provide this flexibility transparently to the customer's application software. The sequence for accomplishing this compliance, according to an exemplary embodiment of the present invention, is schematically illustrated in FIGURE 7A, which schematically illustrates the sequence 310 for the callback mode, and in FIGURE 7B, which schematically illustrates the sequence 320 for the full message mode. The sequence charts present the interaction between the Open Platform API (OPA) 316, a Plug-In (or Utility) 314 and the Application 312 requesting a service.

1. The client application will receive the execution control from OPA at start up. It might also optionally be required to initialize the plug-in service according to the same paradigms that is used for initializing the OPA services (the platform assembly functionality). When the plug-in interface "IY" is called (i.e. a service from the Plugin is requested by the application), the plug-in will rely on one or several OPA service interfaces "IX" to carry out the task. As

noted in FIGURES 7A and 7B, these service request steps are identical for both the callback mode and the full message mode.

2. Depending on how the customer application would like to receive the result, it either returns execution control to OPA (callback mode 310) or polls for messages via full message mode 320 as indicated by 318 in FIGURES 7A and 7B..

3. The platform assembly knows that the service "IX" is carried out for a plug-in and will return the result to the plug-in by invoking a Plugin method e.g. the Handle Message method according to 322. The plug-in will convert the result from the platform assembly into a message format as indicated in FIGURE 7A and return this message to the platform assembly represented by OPA 316 in this case. Note that this step might be omitted if the application has indicated to the plug-in that it only wants to receive messages in callback mode, which will speed up the handling and delivery of the result. The platform assembly is aware of the message mode in which the application waits for the result and will return the result accordingly. Depending on which mode that is active, at least two scenarios are possible:

- If the application is in callback mode 310 according to FIGURE 7A, the result will be given in the callback method "ICBY_OnService". In this case the plug-in will process and format the result so it can be returned via the stack using a procedure call.

  If the application is in full message mode 320 according to FIGURE 7B, the result will be delivered using the message created

by the plug-in in step Handle_Message 322. When the plug-in returns the result in message format to OPA 316, OPA will in turn return control from Get_Message to the application 312, thereby delivering the result.

With the plug-in model according to principles of the present invention, the issues described above with respect to changing the functionality of a mobile terminal platform are overcome. For example, by using OPA plug-ins for the implementation of higher level functionality, the platform provider can offer a slimmer and more stable platform, which still includes the basic and/or default platform functionality that is considered to cover the needs of end user products. This brings the administration of the higher level functionality closer to the application domain and to manufacturers of end-user products and other customers, since the non-standard add-on functionality is provided as a plug-in, which can be easily added/removed depending on the specific product concerned.

In addition, with embodiments of the present invention, third party developers are able to implement add-on functionality as one or several plug-ins, without it being necessary to reveal proprietary information relating to the platform code base to third party developers or others. In particular, since the plug-in is an entity of the application domain software, a third party developer can choose to use only the default/basic platform API.

By exploiting the plug-in technique as described above, a provider of the platform assembly can choose to expose only a controlled and dedicated fraction of the code base to a customer, i.e., one or several of the stand alone plug-ins. The provider may, for example, choose to provide the plug-in as source code to the customer, who can then change and modify the code of the plug-in in order to tailor the functionality and behavior of the plug-in.

By supplying a plurality of plug-ins with a higher level of functionality, it is easy for an end user product manufacturer or other party to extend the functionality of the platform assembly. It is also possible for a platform provider to deliver manufacturer-specific extensions to the platform assembly that is not considered to be part of the basic functionality offering of the platform assembly.

By supplying a plurality of plug-ins with extended functionality, it is also easy to perform a "last minute" configuration of the platform assembly, e.g., to tailor the platform assembly to different needs arising from different products, or to customize the end-user product at the time of purchase. With embodiments of the present invention, it will be easy for end user manufacturers using the platform assembly to make late decisions about whether to use platform provider plug-ins or to develop functionality in-house.

As mentioned above, one example where a higher level functionality of a product platform might be desired is with respect to product-specific functionality such as functionality that controls the look and feel of a user interface. For mobile terminals and other specialized consumer products with advanced user interfaces, the look and feel of the user interface is an important key to differentiation and positioning of the products in the marketplace; and, consequently, is important to commercial success. It is, accordingly, desirable that a platform for mobile terminals and other products provide a great deal of freedom in defining the look and feel of the products.

The separation of a windowing system core from the definition of look and feel is a proven and widely-used concept. The X window system exhibits this, and there are various examples of how systems with very different look and feel can be built on top of the same windowing system using this type of architecture. On the other hand, the ability to alter the look and feel of a given implementation is limited.

Porting of a Java VM is traditionally done at a low level of UI support. The look and feel is, to a high degree, defined by the implementation of graphical objects in Java. The introduction of MIDP (Mobile Information Device Profile) suggests another approach, where graphical objects are only defined at an abstract level. The look and feel is defined by the underlying system on which the VM is executing. If look and feel conformance between Java applications and other applications, executing outside the Java VM, is to be achieved, the same set of graphical objects can be used for both. This is straightforward for a system with a defined look and feel.

A Java VM in a mobile device ideally relies on the platform on which it is running to provide UI support according to the MIDP specification. This means that for a mobile platform including a Java VM, the MIDP support should also be part of the platform. On the other hand, for a platform intended for highly profiled consumer products from different vendors, it is important to provide the highest possible level of control over look and feel. Clearly, the highest level of control is achieved through control of the definition of the graphical objects. These two conflicting requirements on the platform cannot be satisfied with the previously described approaches.

In accordance with a further exemplary embodiment of the present invention, however, a set of graphical objects and utilities are defined as one or more plug-ins in the Open Platform API (OPA). The interfaces to the one or more plug-ins have to be fixed and sufficient for MIDP support, but the implementation may be altered freely.

In accordance with principles of the invention, the implementation of high level graphical objects and utilities can be shared between native applications and execution environments like Java. At the same time, the platform customer's control of the appearance and behavior of the graphical user interface is very high.

While what has been described constitute exemplary embodiments of the invention, it should be understood that the invention can be varied in many ways without departing from the scope thereof. For example, although the present invention has been described primarily in connection with a particular mobile terminal platform assembly, it is not intended to so limit the invention as the invention may also be used in other platforms for mobile terminals and other products. Because the invention can be varied in many ways, it should be recognized that the invention should be limited only insofar as is required by the scope of the following claims.